# Learning without Data: Physics-Informed Neural Networks for Fast Time-Domain Simulation

Jochen Stiasny, Samuel Chevalier, and Spyros Chatzivasileiadis
Department of Electrical Engineering
Technical University of Denmark
Kgs. Lyngby, Denmark
{jbest,schev,spchatz}@elektro.dtu.dk

Abstract—In order to drastically reduce the heavy computational burden associated with time-domain simulations, this paper introduces a Physics-Informed Neural Network (PINN) to directly learn the solutions of power system dynamics. In contrast to the limitations of classical model order reduction approaches, commonly used to accelerate time-domain simulations, PINNs can universally approximate any continuous function with an arbitrary degree of accuracy. One of the novelties of this paper is that we avoid the need for any training data. We achieve this by incorporating the governing differential equations and an implicit Runge-Kutta (RK) integration scheme directly into the training process of the PINN; through this approach, PINNs can predict the trajectory of a dynamical power system at any discrete time step. The resulting Runge-Kutta-based physics-informed neural networks (RK-PINNs) can yield up to 100 times faster evaluations of the dynamics compared to standard time-domain simulations. We demonstrate the methodology on a single-machine infinite bus system governed by the swing equation. We show that RK-PINNs can accurately and quickly predict the solution trajectories.

Index Terms—Runge-Kutta, neural networks, time-domain simulation, transient stability analysis

## I. INTRODUCTION

Time-domain simulation is a fundamental tool for ensuring operational stability of modern electrical power systems. Over the last many decades, power system engineers have developed a mature catalogue of physics-based modeling strategies, e.g., [1], [2], and numerically efficient simulation platforms, e.g., [3], [4]. Building on these advances, emerging synthetic test case libraries [5], [6] are allowing researchers to explore the dynamical complexity of massive power system models which contain tens of thousands of buses.

As grid dynamics become both faster and more distributed, however, the computational expense associated with running full-scale power system simulations is growing considerably. Traditionally, power system engineers have relied on various Model Order Reduction (MOR) methodologies [7]–[9] and modal analysis techniques [10], [11] to overcome the computational burden and complexity of large models. Accordingly, dynamic model reduction tools are built into a variety of commercial simulators (e.g., DIgSILENT, DYNRED and PSS/E) [12] and are commonly used by industry.

While such equivalency tools can produce compact surrogate models, standard MOR has significant drawbacks in certain applications. For example, the majority of MOR techniques used in power systems are only applicable for

This work is supported by the multiDC project funded by Innovation Fund Denmark, Grant No. 6154-00020B, and by the ERC Project VeriPhIED, funded by the European Research Council, Grant Agreement No. 949899

linear systems (e.g., prony analysis and matrix pencil methods [10], [11]; Autoregressive models [13]; Gramiam-based approaches [8]). Second, most MOR methods are projection based, and the resulting dynamical model must still be directly simulated by an ODE solver. Finally, most classical MOR tools which are applicable to power systems cannot efficiently compress any given nonlinear function with an arbitrary degree of accuracy. For example, the Koopman Mode Decomposition requires infinite terms to approximate a logistic map [14], and Sparse Identification of Nonlinear Dynamics (SINDY) [15] may fail if the nonlinear basis functions are poorly chosen.

In contrast to that, Neural Networks (NNs) have the capacity to universally approximate any continuous function with an arbitrary degree of accuracy [16]. Therefore, they are able to overcome all of the aforementioned drawbacks, and they have become a popular alternative to classical MOR approaches. Recently, the so-called Physics-Informed Neural Network (PINN) [17] was proposed as a framework for directly mapping dynamical system inputs (initial conditions) to system outputs (state trajectories). PINNs naturally allow for a direct regularisation of the training process with physical sensitivities, and they also bypass the need for a numerical solver altogether, thus approximating the solution of the underlying ODE system. In [17], continuous and discrete time PINN approaches were presented. In the discrete time formulation, the NN predicts the numerical values associated with an implicit Runge Kutta (RK) integration scheme. The training process of this NN, subsequently referred to as RK-PINNs, does not require any simulated data; hence, it naturally yields a "data free" framework.

Physics-informed neural networks have been first introduced in power systems in our previous work [18], and, since then, have extended in applications related to system identification [19], the transient response of interconnected systems [20], and DC optimal power flow [21]; along the same lines, sensitivity-informed NNs have been recently introduced for AC power flow optimization [22] and physics-informed graphical NNs for parameter estimation [23]. All of these works, however, have utilised a continuous formulation of the problem's underlying physics and have thus required simulated training data.

Given (i) the increasing computational complexity of timedomain integration approaches, (ii) the potential inadequacy of classical MOR tools to approximate all newly emerging non-linearities, e.g., related to converter-dominated systems, and (iii) the emerging success of PINN modeling for power systems, this paper leverages the RK-PINNs framework in order to learn power system dynamics with a ML model. The resulting model can be used, e.g., to screen transient stability contingencies orders of magnitude faster than conventional numerical integration schemes, such as Runge-Kutta, and most attractively, the RK-PINN model parameters can be learned without the need for any simulated power system training data. Accordingly, our contribution lies in

- introducing, for the first time, the RK-PINNs learning framework in power systems applications;
- extending the fundamental structure of RK-PINNs to incorporate variable time steps;
- introducing an additional regularisation term on the RK-PINN's prediction based on the differential equations;
- and providing a publicly available code base.

After introducing the methodology in Section II, we describe the case study in Section III and the results in Section IV. We conclude in Section V.

#### II. METHODOLOGY

We first describe the two fundamental elements of the method, i.e., implicit Runge-Kutta (RK) integration schemes and neural networks. Then, we show how neural networks can assist in solving the non-linear system of equations stemming from the RK scheme. As the methodology is not limited to a specific problem, we introduce it in a general form in this section, and then we show its application to power systems in Section III.

## A. Runge-Kutta schemes

RK methods allow us to find an approximation of the temporal evolution of a dynamical system described by a set of ordinary differential equations (ODEs)

$$\frac{d}{dt}\mathbf{x} = \mathbf{f}(t, \mathbf{x}(t); \mathbf{u}). \tag{1}$$

In (1), x represents the state vector that evolves over time t and u represents control inputs to the system. Function f describes the parametrised update rule for x. These systems of ODEs can be solved by the following general RK scheme [17] for a time step  $\Delta t$ :

$$\boldsymbol{h}^{k} = \boldsymbol{f} \left( t_{0} + \gamma^{k} \Delta t, \boldsymbol{x}^{0} + \Delta t \sum_{l=1}^{s} \alpha^{kl} \boldsymbol{h}^{l}; \boldsymbol{u} \right), \ k = 1, \dots, s,$$
(2)

$$\boldsymbol{x}^1 = \boldsymbol{x}^0 + \Delta t \sum_{k=1}^s \beta^k \boldsymbol{h}^k. \tag{3}$$

The s RK-stages  $h^k$  represent state update vectors according to (2). The prediction  $x^1$  of the system state at time  $t_0 + \Delta t$  can afterwards be calculated by weighing these state update vectors  $h^k$  and applying it for a  $\Delta t$  onto the initial condition  $x^0$  as shown in (3). The properties of the integration scheme largely depend on the coefficients  $\alpha^{kl}$ ,  $\beta^k$ , and  $\gamma^k$ . Their values determine whether the scheme is explicit or implicit and the order of the scheme which governs the truncation error. If the coefficient matrix  $\alpha^{kl}$  has a strictly lower-triangular shape, the scheme is explicit; otherwise, it contains implicit

functions of  $h^k$ . Typical RK-schemes include the forward and backward Euler, the common RK-45 scheme, the trapezoidal method, and many more regularly used schemes. Implicit schemes find use mostly in stiff ODEs as explicit schemes would require a very small time step size due their numerical stability properties. However, the resulting system of nonlinear equations can be difficult to solve, in particular for large number of stages s and systems with many states.

#### B. Neural networks

The reason to use neural networks (NNs) for this problem boils down to exploiting their capacity to approximate the solution to the system of non-linear equations that (2) yields with an arbitrary degree of accuracy, while providing extremely quick evaluations. The NN training problem takes the form

$$\min_{\mathbf{W}, \mathbf{h}} \quad \mathcal{L} \tag{4}$$

$$z_{k+1} = \sigma(W_{k+1}z_k + b_{k+1}), \forall k = 0, 1, ..., K-1$$
 (5)

$$y = W_{K+1} z_K + b_{K+1}. (6)$$

 $W_i$  and  $b_i$  represent the adjustable parameters, namely the weight matrix and bias vector of the i-th layer. Equation (5) describes the hidden layers of the NN in which the input to the layer  $z_i$  undergoes a linear transformation followed by an element-wise non-linearity, e.g., t-anh, to yield the output of the layer  $z_{i+1}$ . The neural network consists of K hidden layers.  $z_0$  equals the input vector and (6) describes the final layer in which we only apply a linear transformation to obtain the NN's ouput y. We will address the formulation of  $z_0$ , y, and the objective  $\mathcal{L}$  in (4) in the following subsections where we incorporate an implicit RK-scheme into this general NN. In this context, we will make use of the tool of automatic differentiation (AD) that allows us to evaluate the derivative of the output variables with respect to the NN's inputs.

# C. Fixed time step RK-PINN

In a first step to incorporate the RK-scheme into NNs, similar to [17], we use the NN to predict the RK-stages for a fixed time step  $\Delta t$  from which we then construct the state prediction  $\hat{x}^1$ :

$$\boldsymbol{z}_0 = [\boldsymbol{x}^0, \boldsymbol{u}] \tag{7}$$

$$\mathbf{y} = [\hat{\mathbf{h}}^{1\top}, \dots, \hat{\mathbf{h}}^{s\top}]^{\top}$$
(8)

$$\hat{\boldsymbol{x}}^1 = \boldsymbol{x}^0 + \Delta t \sum_{k=1}^s \beta^k \hat{\boldsymbol{h}}^k. \tag{9}$$

For all quantities that are based on a NN prediction, we will subsequently use the hat symbol  $(\hat{\cdot})$ . In a supervised learning setting, we would be required to know the values for  $h^i$  in order to adjust the NN's parameters accordingly. By making use of the fact that we obtain a "correct" solution if (2) is satisfied, we can test how well the NN's predictions  $\hat{h}^i$  match the equations and define the vector  $\epsilon^k$  for each RK-stage via

$$\boldsymbol{\epsilon}^{k}(\boldsymbol{x}^{0}, \boldsymbol{u}) = \hat{\boldsymbol{h}}^{k} - \boldsymbol{f}\left(t_{0} + \gamma^{k}\Delta t, \boldsymbol{x}^{0} + \Delta t \sum_{l=1}^{s} \alpha^{kl} \hat{\boldsymbol{h}}^{l}; \boldsymbol{u}\right). \tag{10}$$

Based on (10), we can evaluate the error element-wise  $\epsilon_i^k$  for each RK-stage and across N collocation points (indexed by subscript j)

$$\mathcal{L}_i^k = \sum_{i=1}^N \epsilon_i^k(\boldsymbol{x}_j^0, \boldsymbol{u}_j)^2$$
 (11)

and formulate the training problem as

$$\min_{\boldsymbol{W}_{i},\boldsymbol{b}_{i}} \quad \sum_{i,k} \mathcal{L}_{i}^{k} \tag{12}$$

s.t. 
$$(5), (6), (7), (8)$$
.  $(13)$ 

Note, in the entire calculation, we are not required to know the "true" values of  $h^k$  or  $x^1$ , and hence, there is no need for running simulations to create a dataset of results for  $h^k$  or  $x^1$ . Instead, we purely evaluate  $\epsilon^k$  on a number of values for initial points  $x^0$  and inputs u, i.e., the collocation points.

# D. Variable time step RK-PINN

In the previous subsection, as well as in [17],  $\Delta t$  was a fixed value, and therefore, we would need to train different RK-PINNs if we wanted to incorporate different time steps. This paper extends the method by training a single NN, where  $\Delta t$  is introduced as an input variable:

$$\boldsymbol{z}_0 = [\Delta t, \boldsymbol{x}^0, \boldsymbol{u}]. \tag{14}$$

The only other changes compared to the previous RK-PINN concern the error calculation of the RK-stages. We need to adjust  $\epsilon^k$  to  $\epsilon^k(\Delta t_j, \boldsymbol{x}_j^0, \boldsymbol{u}_j)$  in (10) and (11) since  $\Delta t$  has become an input:

$$\mathcal{L}_i^k = \sum_{j=1}^N \epsilon_i^k (\Delta t_j, \boldsymbol{x}_j^0, \boldsymbol{u}_j)^2.$$
 (15)

We usually think of the implicit RK scheme as a discrete solver that is particularly useful for solving ODEs at a specific time step. By introducing the variable time steps, the RK-PINNs recover the characteristics of a continuous solution approximation such as with other PINNs. This in turn allows us to use the same regularisation which is commonly used with continuous time PINNs [18], [20], in which we evaluate the consistency of the NN sensitivity with the governing differential equations. This error  $\xi$  is calculated by applying AD to  $\hat{x}^1$  and the comparing it with the state update in (1) when using the predicted state  $\hat{x}^1$  as an input:

$$\boldsymbol{\xi}(\Delta t, \boldsymbol{x}^{0}, \boldsymbol{u}) = \frac{\partial}{\partial \Delta t} \hat{\boldsymbol{x}}^{1} - \boldsymbol{f}\left(t_{0} + \Delta t, \hat{\boldsymbol{x}}^{1}; \boldsymbol{u}\right). \tag{16}$$

Analogous to (15), we define a loss term  $\mathcal{L}_i^{dt}$  across the training dataset

$$\mathcal{L}_i^{dt} = \sum_{j=1}^N \boldsymbol{\xi}(\Delta t_j, \boldsymbol{x}_j^0, \boldsymbol{u}_j)^2. \tag{17}$$

This additional loss term still does not require any simulated data, but it places an additional regularisation on the neural

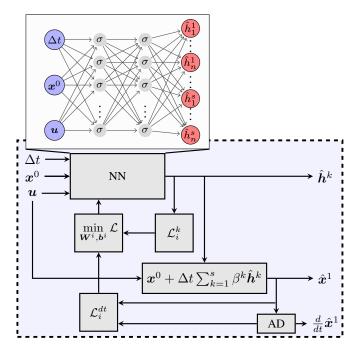


Fig. 1. Architecture of the RK-PINN.

network training procedure. The following optimisation problem describes the training setup in its final form:

$$\min_{\boldsymbol{W}_{i},\boldsymbol{b}_{i}} \quad \sum_{i} \lambda_{i}^{k} \mathcal{L}_{i}^{k} + \sum_{i} \lambda_{i}^{dt} \mathcal{L}_{i}^{dt} \tag{18}$$

s.t. 
$$(5), (6), (8), (9), (14)$$
.  $(19)$ 

The coefficients  $\lambda_i^k$  and  $\lambda_i^{dt}$  serve to balance the influence of the different loss terms on the adjustment of the NN parameters  $W_i$  and  $b_i$ . This prevents a single loss term from dominating the optimisation problem, as this can lead to poor training characteristics.

#### III. CASE STUDY

We demonstrate the proposed methodology on a Single-Machine Infinite-Bus (SMIB) system, which we detail below. This section also specifies the NN parameters and the training setup.

## A. Single-Machine Infinite-Bus system

The SMIB system shown in Fig. 2 represents a second-order generator model connected to an external stiff grid. The voltage angle at the point of connection (Bus 2) is considered the reference angle; it is set constant and equal to  $\delta_{ext}=0\,\mathrm{rad}$ . The state space equations (20) are time-invariant and assume a time-invariant active power production P. The system is furthermore parametrised by the machine's inertia constant  $m=0.4\,\mathrm{p.u.}$ , a damping coefficient  $d=0.15\,\mathrm{p.u.}$ , and the network parameters  $B_{12}=0.2\,\mathrm{p.u.}$  and voltages  $V_1=V_2=1\,\mathrm{p.u.}$ .

$$\frac{d}{dt} \begin{bmatrix} \delta \\ \omega \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{d}{m} \end{bmatrix} \begin{bmatrix} \delta \\ \omega \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} (P - V_1 V_2 B_{12} \sin \delta) \end{bmatrix}$$
(20)

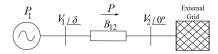


Fig. 2. Single Machine Infinite Bus system

We train the NN on the following domain  $(x^0 = \begin{bmatrix} \delta^0 & \omega^0 \end{bmatrix}^\top)$ :

$$t \in [0, 10]s$$
 (21a)

$$P \in [0, 0.2]$$
 p.u. (21b)

$$\delta^0 \in [-\frac{\pi}{2}, \frac{\pi}{2}] \text{rad} \tag{21c}$$

$$\omega^0 = 0.1 \frac{\text{rad}}{\text{s}}.\tag{21d}$$

#### B. NN training

For the training process, we use a single hidden layer K=1with 50 neurons and the tanh activation function. We test the approach for different numbers of RK-stages  $s \in [4, 8, 16, 32]$ . Our implementation utilises the TensorFlow framework [24] and it is publicly available on github<sup>1</sup>. The training problem (18) is solved by using the stochastic gradient descent method Adam [25] with a decaying learning rate of  $0.05 \cdot 0.995^{\frac{L}{100}}$ . where E is the number of epochs. For the experiments, we first create a large database of points across the input domain with increments of  $0.1\,\mathrm{s},\,0.004\,\mathrm{p.u.}$  and  $\frac{\pi}{50}\,\mathrm{rad}$ , corresponding to (21a), (21b), and (21c), respectively. From this database, we randomly sample  $N \in [50, 100, 200, 1000]$  collocation points, i.e., points that define  $z_0$  but do not include the associated target values for  $h^k$ . We train the models for 100'000 epochs, where an epoch refers to an optimisation step with respect to the loss function  $\mathcal{L}$  in (18) across the collocation points. During the training we monitor the loss function  $\mathcal{L}$  across another 1000 points, which serve as the validation set, and we use it for an early stopping of the training to prevent over-fitting. Lastly, we assess the accuracy of the model by evaluating the prediction error  $e_{\delta}$ :

$$e_{\delta} = \left(\delta^{1}(\Delta t, \delta^{0}, P) - \hat{\delta}^{1}(\Delta t, \delta^{0}, P)\right)^{2}$$
 (22)

with  $\hat{\delta}^1$  from  $\hat{x}^1 = \begin{bmatrix} \hat{\delta}^1 & \hat{\omega}^1 \end{bmatrix}^\top$ . When presenting the results, we will use percentiles of the distribution of  $e_{\delta}$  to describe its properties across a dataset. The k-th percentile refers to the value below which k % of the distribution lie.

#### IV. RESULTS

First, we will demonstrate the computational advantages of RK-PINNs for the evaluation of ODEs. Second, we show the high quality of the achieved accuracy for different RK-PINNs and point out the error characteristics of the method.

## A. Computational advantage in evaluation

The primary advantage of using NNs for approximating the solution to ODEs lies in the speed of the evaluation. Figure 3 illustrates the enormous computational advantage of RK-PINNs. The most powerful feature of RK-PINNs is

<sup>1</sup>Github repository: github.com/jbesty

that the evaluation time is independent of the time step  $\Delta t$  whereas the traditionally applied methods suffer from increasing computational effort with larger time steps, e.g., when we wish to determine the state of the system at  $\Delta t = 1s$ or  $\Delta t = 10s$ . We show this trend for the implementation of full implicit RK (IRK) schemes with s=4 and s=32RK-stages, respectively. For reference, we also use an implicit 'Radau'-scheme and the explicit 'RK-45'-scheme as provided in the scipy.optimize package. It is worth pointing out that increasing values for  $\Delta t$  pose a challenge for lower order implicit RK schemes in terms of convergence. For example, IRK 4 in Fig. 3 failed to converge beyond  $\Delta t = 2s$ . This motivates either moving to higher order and hence more expensive implicit RK schemes or using adaptive solution approaches. In contrast, RK-PINNs do not suffer from convergence issues; and as we will see in Section IV-C, they achieve acceptable accuracy even with few RK-stages. As far as the computational cost of RK-PINNs is concerned, in contrast to the conventional approaches, this is primarily governed by the NN's size whereas the number of RK-stages plays a minor role. Since the involved calculations entail only a matrix multiplication and a non-linear function evaluation per NN layer, it is apparent that even a much larger RK-PINN would still retain a significant computational advantage over the other methods. To illustrate that, in Fig. 3, we compare a PINN with 3 layers and 500 neurons each, labelled PINN 4\*, with our standard PINNs of 1 layer with 50 neurons, labelled PINN 4 and PINN 32.

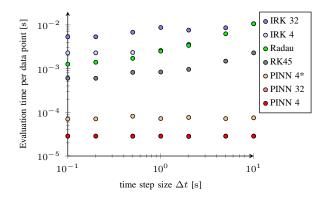


Fig. 3. Evaluation time of different methods for increasing time step sizes. The legend indicates the method and the order. We note that the evaluation time difference between PINN 4 and PINN 32 is visually indistinguishable.

It is important to note that Fig. 3 does not indicate anything about the accuracy of the RK-PINNs. For the other methods, the evaluation time reports the time required to reach a certain tolerance, in this case  $10^{-13}$ , for which each method will achieve a certain accuracy if it converges. Lowering numerical tolerance requirements can speed up the solution time at the risk of obtaining inaccurate solutions. For RK-PINNs, in contrast, the achieved accuracy is dependent on two aspects: i) the NN size as previously mentioned and ii) the training procedure. Hence, RK-PINNs of exactly the same size can have nearly identical evaluation times but achieve vastly different accuracy levels.

In essence, RK-PINNs, and in fact any similar NN architecture, will be much faster to evaluate than traditional ODE

solvers; however, to exploit this advantage we have to show sufficient levels of accuracy. We explore this in the following subsections.

### B. Evaluating and interpreting accuracy

The main metric of interest is the achieved accuracy across the input domain defined in (21). To give a first impression of the accuracy that we can accomplish with RK-PINNs, we consider three trajectories of  $\delta$  in Fig. 4. The maximum error  $e_{\delta}$  evaluates to  $1.0 \times 10^{-2}$  for the red,  $5.0 \times 10^{-3}$  for the yellow, and  $1.0 \times 10^{-3}$  for the blue trajectory.

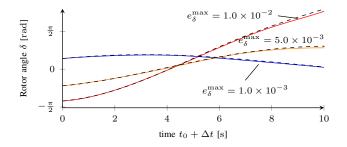


Fig. 4. Prediction (coloured) and ground truth (black dashed) for trajectories

Before comparing numeric metrics, it is worth taking a look at the distribution of the  $e_{\delta}$  across the test dataset to give an understanding of how we arrive at the subsequent numbers. Figure 5 depicts  $e_{\delta}$  ordered by magnitude, i.e., the corresponding percentile is shown on the x-axis. Each grey line represents this error distribution for a trained RK-PINN. The different curves arise because of the random initialisation of the NN's weights and biases and the random sampling of the collocation points. We subsequently calculate the mean (shown in red in Fig. 5) and standard deviation of the percentile values to report the results in Table I and Table II.

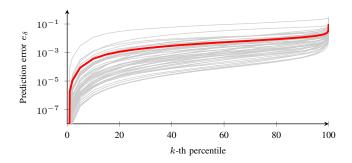


Fig. 5. Each grey line represents the error distribution on the test dataset of a trained NN. The red solid line is the mean of these error curves.

# C. The effect of the number of RK-stages on accuracy

Table I lists the results of the main experiment where we compare  $e_{\delta}$  for RK-PINNs with different numbers of RK stages s. To be clear, a 4-stage RK-PINN outputs 4 RK state update prediction vectors  $\hat{h}^1, \ldots, \hat{h}^4$ , while a 32-stage RK-PINN outputs 32 such prediction vectors. The main insight is that for the given training setup more RK-stages s improve both the mean and the standard deviation across the various training runs. This trend is consistently observable

across all percentiles. This prediction accuracy must not be confused with the truncation error that reduces with higher order schemes; the present effect is the result of better approximations of  $h^k$ . At this point, we point out once again that these results have purely been obtained by using the implicit RK-scheme and no simulation had to be performed in advance. The drawback from this are longer training times (in the order of tens of minutes), especially for higher-order systems.

TABLE I MEAN AND SD OF THE k-TH PERCENTILE OF  $e_\delta$  FOR DIFFERENT IMPLICIT RK SCHEMES IMPLEMENTED IN RK-PINNS

RK	k-th percentile					
s	100	90	50	10		
4	$1.33 \pm 0.96$	$1.81 \pm 2.68$	$6.29 \pm 12.1$	$4.47\pm11.6$		
8	$1.00\pm0.70$	$1.17 \pm 2.57$	$4.04 \pm 11.3$	$3.67 \pm 16.8$		
16	$0.65 \pm 0.45$	$0.81 \pm 1.01$	$2.53 \pm 4.03$	$1.52 \pm 2.81$		
32	$0.58 \pm 0.48$	$0.35 \pm 0.63$	$0.97 \pm 2.26$	$0.60 \pm 2.31$		
	$\times 10^{-1}$	$\times 10^{-2}$	$\times 10^{-3}$	$\times 10^{-4}$		

## D. The effect of the number of collocation points on accuracy

Beside the number of RK-stages, we investigate how the number of collocation points affects the accuracy. Table II presents the results and surprisingly, the outcome is not as clear as for the RK-stages. Whereas more points clearly lead to the smallest maximum errors (100th-percentile), fewer collocation points perform consistently better on lower percentiles. The answer lies in the error characteristics and the training process. If we consider the error across the test dataset for a

TABLE II MEAN AND SD OF THE k-th percentile of  $e_\delta$  for different numbers of collocation points N

	k-th percentile					
N	100	90	50	10		
50	$14.2 \pm 15.7$	$3.01 \pm 1.57$	$2.31 \pm 1.27$	$0.49 \pm 0.36$		
100	$5.75 \pm 6.74$	$1.09 \pm 0.73$	$1.14 \pm 1.10$	$0.29 \pm 0.45$		
200	$2.21 \pm 1.76$	$0.53 \pm 0.49$	$1.03 \pm 1.89$	$0.30 \pm 0.62$		
1000	$1.33 \pm 0.96$	$1.81 \pm 2.68$	$6.29 \pm 12.1$	$4.47 \pm 11.6$		
	$\times 10^{-1}$	$\times 10^{-2}$	$\times 10^{-3}$	$\times 10^{-4}$		

single trained NN, we observe a fairly consistent distribution across each dimension in the input domain. This is shown in the subplots in Fig. 6 for a case with only 50 collocation points on the left, and a case with 1000 collocation points on the right. The different shading indicates the bands in which 100%, 90%, 50% of the error lies and the black line represents the median. These results suggest that, generally speaking, the RK-PINNs exhibit good generalisation abilities. However, more collocation points yield smoother and narrower yet worse approximations, except for the maximum error as it was indicated in Table II. The less extreme errors can be explained by the fact that more collocation points allow better approximations at the input domain boundaries, a common phenomenon in NN training. Figure 6 shows this behaviour,

except for small time steps where all  $\hat{\boldsymbol{h}}^k$  should tend to 0, hence, potential approximation errors are usually small. The reason for the slightly worse overall predictions with N=1000 remains to be investigated, but it seems that more collocation points complicate the training process. Eventually, a trade-off between potentially longer training times and the danger of larger errors has to be found.

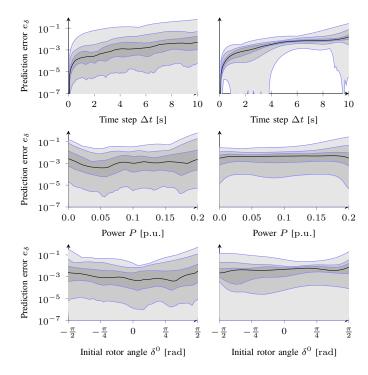


Fig. 6. Distribution of the prediction error  $e_{\delta}$  across the three input dimensions  $(\Delta t, P, \delta^0)$  for a network with s=4 and N=50 (left column) and N=1000 (right column).

# V. CONCLUSION

In this work, we have introduced an approach, known as RK-PINNs, for solving parameterised power system ODEs. As shown by our results, RK-PINNs are capable of reducing the computation time required by conventional solvers by up to two orders of magnitude. Unlike many conventional MOR methods, RK-PINNs can provide an arbitrary degree of approximation accuracy of any continuous function; thus, their applicability is not limited to any narrow range of problems. Furthermore, our results indicate that RK-PINNs can be trained on, and successfully predict, low order RK integration schemes which conventional numerical solvers fail to solve entirely (given a sufficiently large time step). Given their excellent predictive accuracy, as showcased in this paper, RK-PINNs can thus be used for extremely fast screening of a very large number of potential power system contingencies. Future work will focus on (i) reducing the training time required by RK-PINNs, (ii) developing enhanced approaches for selecting optimally representative collocation points, and (iii) exploring training optimisation approaches which will scale favorably with larger model system models.

#### REFERENCES

- [1] P. Kundur et al., Power system stability and control, vol. 7. McGraw-Hill New York, 1994.
- [2] P. Sauer and M. Pai, Power System Dynamics and Stability. Stipes Publishing L.L.C., 2006.
- [3] F. Milano, "An open source power system analysis toolbox," *IEEE Transactions on Power Systems*, vol. 20, no. 3, pp. 1199–1206, 2005.
- [4] W. Long, D. Cotcher, D. Ruiu, P. Adam, S. Lee, and R. Adapa, "Emtp-a powerful tool for analyzing power system transients," *IEEE Computer Applications in Power*, vol. 3, no. 3, pp. 36–41, 1990.
- [5] T. Xu, A. B. Birchfield, and T. J. Overbye, "Modeling, tuning, and validating system dynamics in synthetic electric grids," *IEEE Transactions on Power Systems*, vol. 33, no. 6, pp. 6501–6509, 2018.
- [6] T. Xu, A. B. Birchfield, K. S. Shetye, and T. J. Overbye, "Creation of synthetic electric grid models for transient stability studies," in *The 10th Bulk Power Systems Dynamics and Control Symposium (IREP 2017)*, pp. 1–6, 2017.
- [7] D. Chaniotis and M. Pai, "Model reduction in power systems using krylov subspace methods," *IEEE Transactions on Power Systems*, vol. 20, no. 2, pp. 888–894, 2005.
- [8] A. Ramirez, A. Mehrizi-Sani, D. Hussein, M. Matar, M. Abdel-Rahman, J. Jesus Chavez, A. Davoudi, and S. Kamalasadan, "Application of balanced realizations for model-order reduction of dynamic power system equivalents," *IEEE Transactions on Power Delivery*, vol. 31, no. 5, pp. 2304–2312, 2016.
- [9] X. Zhang, Y. Xue, S. You, Y. Liu, and Y. Liu, "U.s. eastern interconnection (ei) model reductions using a measurement-based approach," in 2018 IEEE/PES Transmission and Distribution Conference and Exposition, pp. 1–5, April 2018.
- 10] M. Crow and A. Singh, "The matrix pencil for power system modal extraction," *IEEE Transactions on Power Systems*, vol. 20, no. 1, pp. 501–502, 2005.
- [11] D. Trudnowski, J. Johnson, and J. Hauer, "Making prony analysis more accurate using multiple signals," *IEEE Transactions on Power Systems*, vol. 14, no. 1, pp. 226–231, 1999.
- [12] F. Milano and K. Srivastava, "Dynamic rei equivalents for short circuit and transient stability analyses," *Electric Power Systems Research*, vol. 79, no. 6, pp. 878–887, 2009.
  [13] J. Chai, Y. Liu, Y. Liu, N. Bhatt, A. D. Rosso, and E. Farantatos,
- [13] J. Chai, Y. Liu, Y. Liu, N. Bhatt, A. D. Rosso, and E. Farantatos, "Measurement-based system reduction using autoregressive model," in 2016 IEEE/PES Transmission and Distribution Conference and Exposition, pp. 1–5, 2016.
- [14] S. L. Brunton and J. N. Kutz, Data-driven science and engineering: Machine learning, dynamical systems, and control. Cambridge University Press, 2019.
- [15] J.-C. Loiseau, B. R. Noack, and S. L. Brunton, "Sparse reduced-order modelling: sensor-based dynamics to full-state estimation," *Journal of Fluid Mechanics*, vol. 844, p. 459–490, 2018.
- [16] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, no. 6, pp. 861– 867, 1993.
- [17] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.
- [18] G. S. Misyris, A. Venzke, and S. Chatzivasileiadis, "Physics-Informed Neural Networks for Power Systems," in 2020 IEEE Power & Energy Society General Meeting, pp. 1–5, Aug. 2020.
- [19] J. Stiasny, G. S. Misyris, and S. Chatzivasileiadis, "Physics-informed neural networks for non-linear system identification for power system dynamics," in 2021 IEEE Powertech, pp. 1–7, 2021.
- [20] J. Stiasny, G. S. Misyris, and S. Chatzivasileiadis, "Transient Stability Analysis with Physics-Informed Neural Networks," arXiv:2106.13638, June 2021.
- [21] R. Nellikkath and S. Chatzivasileiadis, "Physics-informed neural networks for minimising worst-case violations in dc optimal power flow," 2021.
- [22] M. K. Singh, V. Kekatos, and G. B. Giannakis, "Learning to Solve the AC-OPF using Sensitivity-Informed Deep Neural Networks," arXiv:2103.14779, Mar. 2021.
- [23] L. Pagnier and M. Chertkov, "Physics-Informed Graphical Neural Network for Parameter & State Estimations in Power Systems," arXiv:2102.06349, Feb. 2021.
- [24] M. Abadi et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," arXiv:1603.04467, Mar. 2016.
- [25] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," arXiv:1412.6980, Jan. 2017.